Last updated: Thursday, May 14, 2020

# StreamSDK Documentation (3.2.0)

## Welcome to StreamSDK 3.2.0!

StreamSDK began life as “Video Chat” on the Unity Asset Store in 2013. Since then it has grown up a tremendous amount, especially in 2018 when StreamSDK (Video Chat 2.0) was released.

With its 3.0 release StreamSDK has become free to start using on PC/Mac/Linux. Once you’re comfortable developing with StreamSDK you can upgrade your API Key to support new platforms and a higher number of connections when your app/game becomes popular.

In addition to being free-to-start, StreamSDK 3.0 is also faster (80-100% better framerates), allows for in-app audio streaming (any audio playing inside your app can be transmitted), and all of these new improvements can be fully realized using Photon’s Self-Hosted Server, which can be deployed to any Windows-based PC. StreamSDK will still work with any networking solution or platform you might desire, but Photon’s Self-Hosted Server is our current focus.

Stream SDK 3.2.0 has been built targeting Unity 2019.1.8f1. When working in Unity 2019.3 and above, the “Assets / Photon Unity Networking / Demos” folder may need to be removed from the project since they leverage deprecated (as of 2019.3) GUI code.

See the readme file included in the StreamSDK/Doc folder in Unity for version history.

## Contents

Technical Notation Rules

How to get your API keys

How to upgrade StreamSDK

Before importing StreamSDK

After importing StreamSDK

StreamSDK and Photon Self-Hosted Server

StreamSDK and Photon Self-Hosted Server for WebGL

StreamSDK Component

StreamSDK Advanced Options

StreamSDKTransporter Component

StreamSDKUIManager Component

StreamSDKOptionDropDown and StreamSDKOptionSlider Components

StreamSDKOptionDropDown Component

StreamSDKOptionSlider Component

StreamSDK Getters (Network Transmission)

StreamSDK Getters (Error)

StreamSDK Getters (Input)

StreamSDK Getters (Mouse)

StreamSDK Getters (Usage)

StreamSDK Setters

Setting up with Photon

Calibrating the Mic

Integrating StreamSDK into your Project

Creating the Base3D scene from scratch (almost)

Audio Optimization

## Technical Notation Rules

1. Quotes are also used to signify named objects in the Hierarchy or Project View.
2. Capital words are used to signify Unity Editor windows such as Inspector, Hierarchy, Project View, as well as Unity Editor menu choices.
3. Capital words are also used to signify class and method names in the Inspector or in code.
4. Capital-spaced words are used to signify component field names in the Inspector.
5. Camel case words are used to signify field names as they would be seen in code.
6. Lower case words are used to signify generic objects or terms such as game object, asset, prefab, component, scene, class, method, or field.

## How to get your API Keys

Visit http://www.streamsdk.com and register a new account.

Once registration is complete, log in and visit the Dashboard (which should look similar to the image below). **The first thing to do** once StreamSDK is imported is to locate the StreamSDK Prefab in the Project View under “Assets / StreamSDK / Core” and **set its Api Key and Api Secret** fields to the values from your account on the website.

## How to upgrade StreamSDK

At any point, you can add or remove subscribed platforms or connection limits. Simply visit http://www.streamsdk.com, go to the Dashboard, and click on the “Subscribe” or “Unsubscribe” button associated with the service option you want to control. Subscriptions take a few minutes to show up in the Dashboard.

**Subscriptions** can be **disconnected** through your PayPal account as well.

PayPal.com **> Summary > Settings (Gear Icon) > Account Settings > Money, banks and cards > Set Automatic Payments**

## Before Importing StreamSDK

*Note - It's OK if these steps are done after importing the StreamSDK package too.

Example scenes use multiple cameras with various Culling Mask and Depth settings to separate rendering paths between what is native, what is cloned, what is streamed, and what is received.

**1. Add Layers:**

Clone (Layer 9 in StreamSDK Demos)
StreamDisplay (Layer 10 in StreamSDK Demos)
ContentCanvas (Layer 11 in StreamSDK Demos)

**2. Import "Photon Unity Networking Classic - Free" from the Unity Asset Store.**

StreamSDK can be used with any networking library, however we are currently focused on Photon, and a specific version at that.

See "Setting up with Photon" below.

## After Importing StreamSDK

If Photon is used, then the “StreamSDK / Tool / PhotonWrapper / PhotonWrapper.cs” script must be edited (after Photon is imported).

On line 1 of the script, simply comment out:

#define submit

to

//#define submit

This will automatically bulk comment out the placeholder class (that compiles fine without Photon imported) and unlocks the production class that Photon will actually work with.

## StreamSDK and Photon Self-Hosted Server

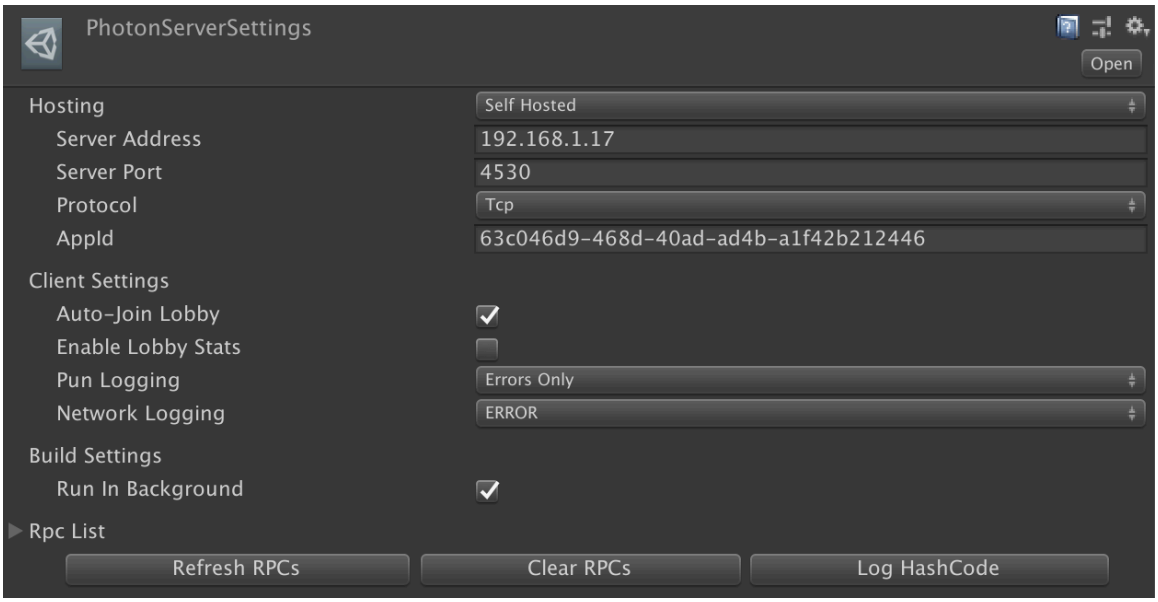
(step-by-step instructions are further down in the document)

*Note - as of StreamSDK 3.0, we HIGHLY recommend using Photon's Self-Hosted Server Solution. Full performance of StreamSDK cannot be realized with Photon Cloud's limitations.

*Note - do not use the AppId pictured here ;)

**https://www.photonengine.com/en-US/Server**

**Figure Below: Photon Self-Hosted Server Typical (Search for "PhotonServerSettings" in Project View to find this asset)**



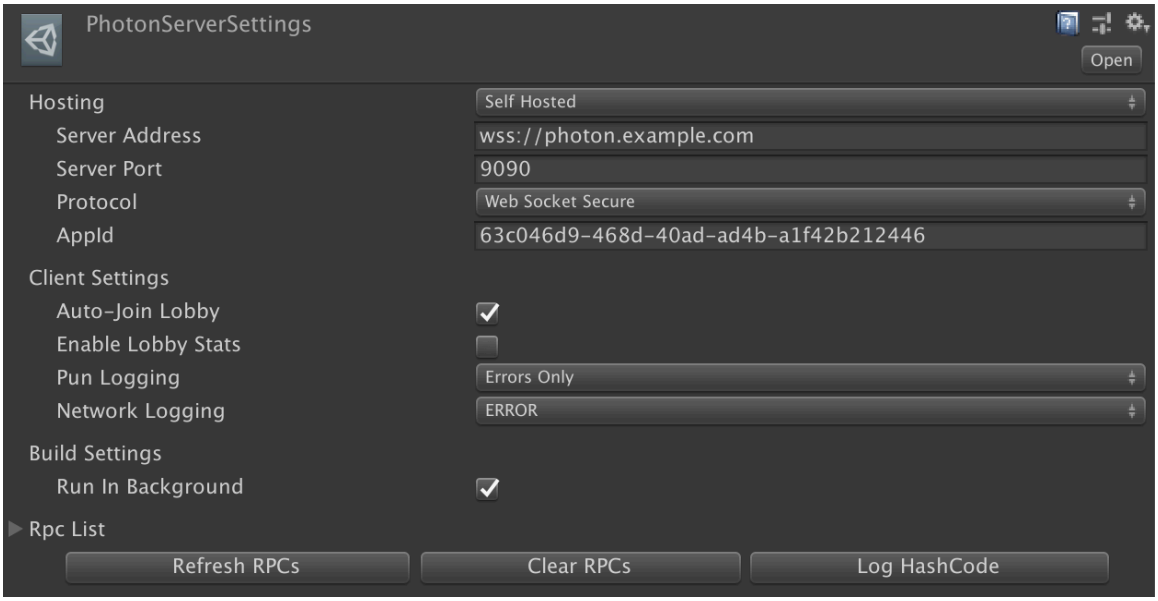
## StreamSDK and Photon Self-Hosted Server for WebGL

If you're using WebGL, you will need to set up a Web Socket Secure server following these instructions:

*Note - do not use the AppId pictured here ;)

**https://doc.photonengine.com/en-us/server/current/operations/websockets-ssl-setup#guide**

**Figure Below: Photon Self-Hosted Using WSS for WebGL Access. (Search for "PhotonServerSettings" in Project View to find this asset)**



## StreamSDK Component

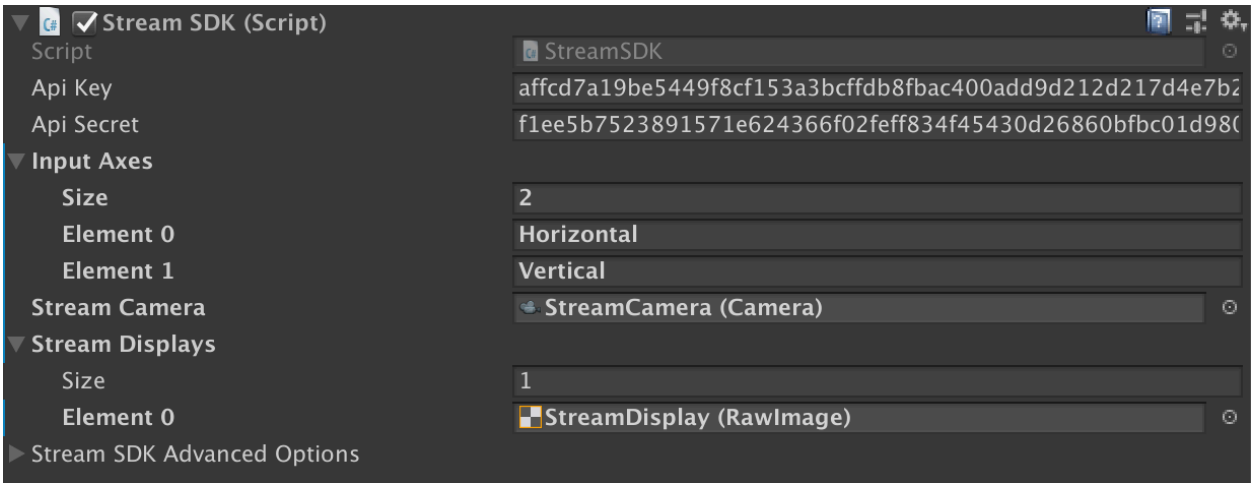
We have tried to keep the StreamSDK component simple, but it has increased in complexity from version 2.0. The additional complexity comes from the API Keys and the Stream SDK Advanced Options features.

*Note - When building for WebGL, drag the “Assets / StreamSDK / Core / WebGL / StreamSDK > StreamSDK” script onto your StreamSDK object, copy/paste the values from the standard version on the new WebGL version of the component, and finally disable the standard version.

*Note - For WebGL functionality, you can also use the StreamSDKWebGL prefab located at: “Assets / StreamSDK / Core / WebGL / StreamSDKWebGL.prefab”

*Note - if the StreamSDK component is selected in the Hierarchy while working in the Unity Editor, audio artifacts will occur. When testing and making changes, be sure to select, change, and deselect (click on empty space in the Hierarchy) to ensure that audio processing occurs properly.

*Note - There should only ever be one AudioListener in your scene and it should be assigned to the StreamSDK game object.




**Api Key and Api Secret** - Your keys, copied and pasted from the StreamSDK website’s Dashboard.

**Input Axes** - An array of input axis names (from the Unity Project Settings > Input manager) to be tracked.

**Stream Camera** - The Camera component in your scene that you want to stream. Whatever the camera “sees” is what will be streamed. So, if the camera is looking at a

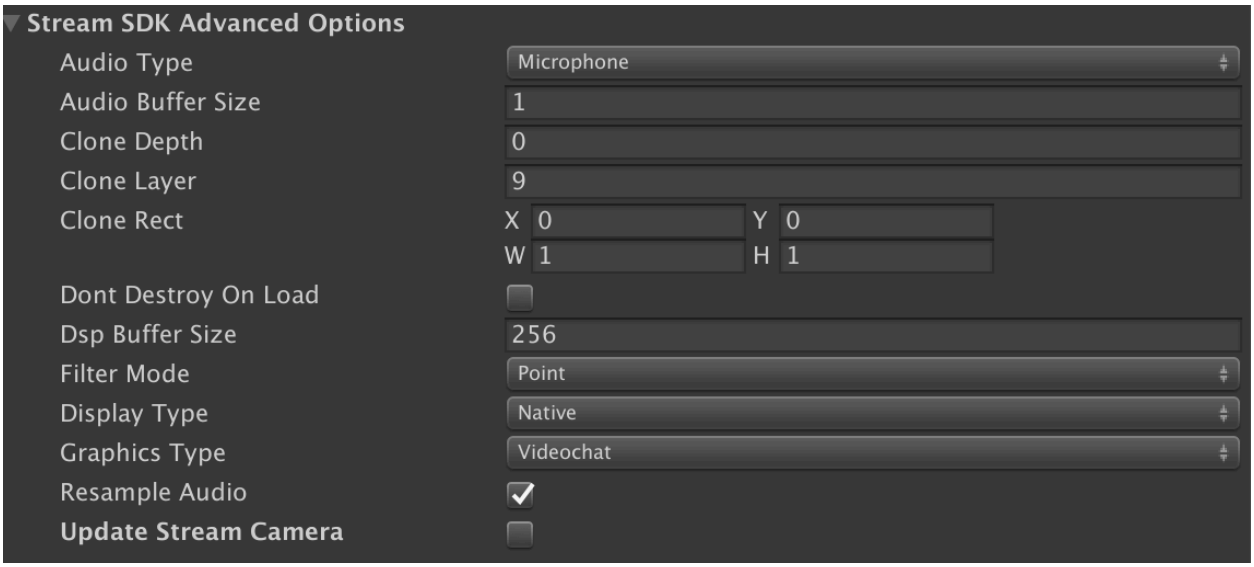
WebCamTexture on a quad, it will stream that, if it's looking at a 3D world it will stream that, etc.

**Stream Displays** - These are the RawImage components in your scene that are set to receive streaming video from other participants. Since StreamSDK supports many-to-many streaming, you can add multiple Stream Displays, one to receive video from each participant in your stream/chat/etc.

---

## StreamSDK Advanced Options

When the Stream SDK Advanced Options expansion is open, an array of new settings are revealed. By default they will yield functionality similar to StreamSDK 2.0. However, when changed these options help StreamSDK 3.0 reach new levels of capability and performance.




**Audio Type - StreamSDK supports 5 Audio Types currently**

- **Microphone**, Audio recorded by the microphone at StreamSDK.instance.micFrequency will be sent from the local machine to connected receivers

- **Microphonemute**, Audio recorded by the microphone at StreamSDK.instance.micFrequency will be sent from the local machine to connected receivers but no audio will play on the local machine, at least not within the Unity app.

- **Micgamemute**, Audio recorded by the microphone at StreamSDK.instance.micFrequency will be played back locally and combined with in-

game music and sound FX before being sent to connected receivers but no audio will play on the local machine, at least not within the Unity app.

- **Game**, All audio played by the app/game will be recorded at StreamSDK.instance.audioFrequency and be sent from the local machine to connected receivers.

- **Gamemute**, All audio played by the app/game will be recorded at StreamSDK.instance.audioFrequency and be sent from the local machine to connected receivers. However, the audio will be silenced on the local machine. Similar to the Blind Display Type, this will only allow the audio to be heard on the receiving end of the transmission (useful for cloud situations etc.)

- **None**, No audio will be sent to connected receivers but audio will still play on the local machine.

- **Nonemute**, No audio will be sent to connected receivers and no audio will play on the local machine either.

**Audio Buffer Size** - By default this is set to 1. However, if receivers are experiencing pauses in audio, this can be increased to 2 (other values might be too high) to close those gaps in audio. Increasing this value also increases the latency of the audio, which buffers before being sent.

**Audio Compressions** - The number of iterations to compress audio data.

**Clone Depth** - This is the depth at which the Clone Layer will be rendered relative to cameras (and their various depths) in the scene.

**Clone Layer** - This is the layer in your scene that you want the cloned Display Type object to appear. In StreamSDK demos “Clone” is typically Layer 9 but it just needs to match whatever layer index is representing the Clone Layer in the project.

**Clone Rect** - This is the normalized viewport rect that the cloned Display Type object will occupy.

**Dont Destroy On Load** - This flag allows StreamSDK to stay alive across multiple scenes, useful when streaming entire games from a server to remote clients.

**DSP Buffer Size** - This int represents the audio latency / quality tradeoff. A DSP Buffer Size of 256 has low latency (a la Unity Editor > Project Settings > Audio > DSP Buffer Size > Best Latency). A size of 1024 is Unity’s “Best Performance”. Values expected are 256, 512, 1024, 2048, 4096, etc. Not all platforms support all values.

**Filter Mode** - This drop down allows StreamSDK to control the filtering mode of video both sent and received.

**Display Type** - StreamSDK supports **3 Display Types** currently

- **Native**, The Stream Camera will perform a native render and the local view will be at the full resolution of your app as set by Unity.
- **Clone**, The Stream Camera will not perform a native render and the local view will be rendered at the same resolution as the stream as set by StreamSDK. This option offers a vast increase in performance but if you're broadcasting a stream at 180p you will also be playing the app/game at 180p.
- **Blind**, Nothing will be rendered on the local machine. This could be useful if you are creating a cloud rendered app/game and do not want to see what is rendered locally i.e. you only want it to be transmitted and viewed on the receiving end of the stream.

**Graphics Type** - StreamSDK supports **5 Graphics Types** currently. This option is currently very limited in version 3.0, but will be expanded upon in upcoming versions.

- **Ar**, Optimized for mobile or head mounted displays rendering AR content.
- **Game 2d**, Optimized for 2D apps/games.
- **Game 3d**, Optimized for typical 3D apps/games.
- **Videochat**, Optimized for Video Chat, with mobile devices in mind.
- **Vr**, Optimized with immersive virtual reality in mind.

**Resample Audio** - StreamSDK will resample audio by default to the audio frequency as set by SetAudioFrequency(). Unchecking this option will stop StreamSDK from resampling audio.

**Turbo Compress** - Video compression may be sped up dramatically by using this option, depending on the platform.

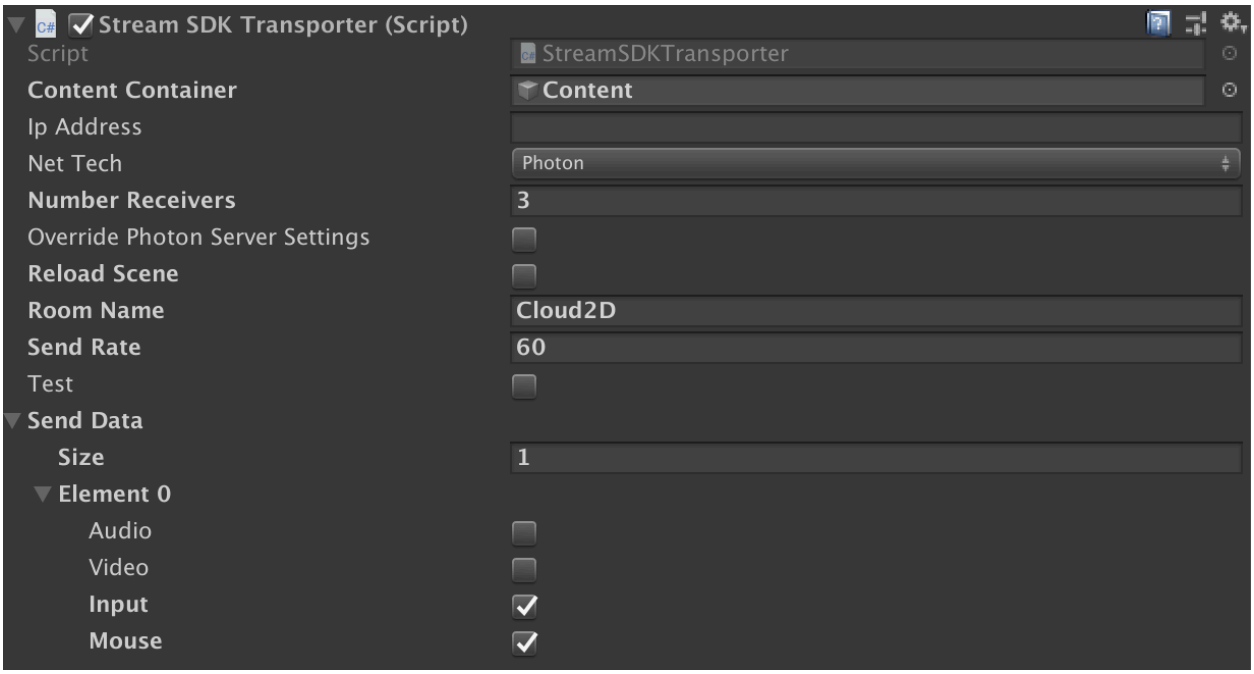
**Turbo Decompress** - Video decompression may be sped up dramatically by using this option, depending on the platform.

**Update Stream Camera** - Checking this option will allow runtime changes to the StreamCamera to be registered with the stream.

**Video Compressions** - The number of iterations to compress video data.

## StreamSDKTransporter Component

As stated above, although our focus is currently on Photon Self-Hosted Server, StreamSDK can be used with any networking library. The StreamSDKTransporter allows you to easily add networking paths without having to change the core interface of network operations. Everything should be handled using StreamSDKTransporter as a wrapper for multiple networking libraries. Internally this has been done successfully with a number of libraries including Photon and TNet.




**Content Container** - Once the network server is connected to and the experience begins, this game object (which presumably contains all the game objects necessary for the experience) will be SetActive( true ). In the Project View, “Assets / StreamSDK / Demo / Car3D” scenes … the car, racetrack, camera to be streamed, etc. are all contained under one top-level object called “Content” in the Hierarchy. This Content Container is set active and all other Containers assigned to the StreamSDKUIManager are set inactive when the race is started (hosted), joined, or tested.

**Ip Address** - The IP Address of the server associated with this networked experience (mainly used for TNet currently, which will be reintroduced in a later version). This may also be useful when Override Photon Server Settings is checked and Photon Self-Hosted Server is used.

**Net Tech** - This can currently be set to Photon or TNet as those are the two networking libraries for which internal wrappers exist (currently the TNet wrapper is not included in StreamSDK 3.0 by default, it will return at a later point). The StreamSDKTransporter can be extended with additional Net Tech options at will.

**Number Receivers** - The number of remote connections to send data to.

**Override Photon Server Settings** - “PhotonServerSettings” is an asset that can be found in the Project View of Unity. It makes it easy to get started but there are times that you may want to control Photon from code. An example can be found in PhotonWrapper, in which the “Shell / MenuCanvas / OptionsNetwork” menu in the Hierarchy (of default StreamSDK demos) controls the Photon Region and Protocol (TCP or UDP) for example. As a side note, TCP is the recommended networking protocol for StreamSDK if you are not using WebGL (which requires Web Socket Secure).

**Reload Scene -** By default, StreamSDK operates without reloading the scene to avoid issues with WebGL. Check this box if scene loading is required and examine the way that StreamSDKTransporter operates upon QuitServer().

**Room Name** - The name of the room on the server to join. Most networking libraries or systems have a room system with accompanying names in order to route connections properly. This should be unique per app/game at least but can go further, for example it could be “GameName” + “UserName”.

**Send Rate** - How many times per second network packets will go out. This can affect performance in many ways. If the Send Rate is too low, audio may lag behind on the receiving end. If the Send Rate is too high, you will lose performance and perhaps tax your backend server (Photon Cloud has packet-per-second limits, one of the many reasons we recommend Photon Self-Hosted Server).

**Test** - This is a bool that determines whether StreamSDK is in Test mode, thus sending data to itself in offline mode. This is set via the StreamSDK.StartServer() call as the second parameter.

**SendData** - This is an array of bools, that allows different data to be routed to different receivers. The indexing is based on the configuration of expected receivers. For example if a game caster is created that streams a game to two players. The game caster will have a SendData size of two each with Audio and Video checked. The game controller (thin client) will have a SendData size of one, with Input checked.

- **Audio,** This is a bool that determines if audio data will be sent to this index of the SendData array.

- **Video,** This is a bool that determines if video data will be sent to this index of the SendData array.

- **Input,** This is a bool that determines if input data (defined by the InputAxes of the StreamSDK component) will be sent to this index of the SendData array.

- **Mouse,** This is a bool that determines if mouse data will be sent to this index of the SendData array.

- **Stream,** This is a bool that determines if all data will be combined into a single packet, reducing the number of packets sent by StreamSDK.

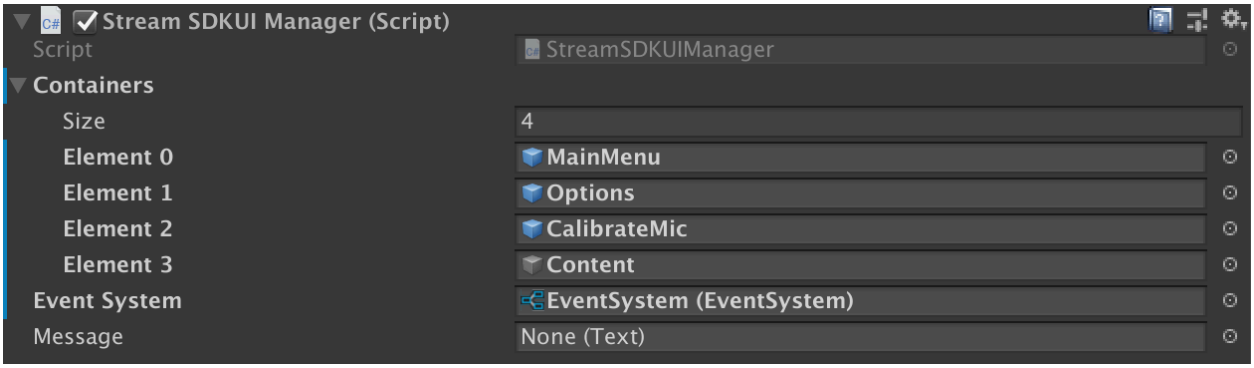
* Note - The Stream option is great for reducing network traffic, however packet size can become an issue when using high quality audio and video simultaneously. Try sending Audio and Video separately (by checking those boxes and not stream) if audio seems to be dropping.

## StreamSDKUIManager Component

The StreamSDKUIManager component allows various containers to be instantly turned off when another is turned on. Simply call StreamSDKUIManager.instance.ShowContainer( GameObject container ) and that container will be SetActive( true ) while all others assigned to the component will be SetActive( false ). This makes it easy to switch between menus or views (for example in the StreamSDK Demos, the “Content” GameObject contains the meat of the game/app/simulation).

The Event System field is required to avoid inputs before initialization of StreamSDK. If the Event System field is left as None, then StreamSDKUIManager will set this field to whatever EventSystem.current is.

*Note - The container at element 0 of the Containers array will be the default container. On Start() StreamSDKUIManager will call ShowContainer( containers[ 0 ] ). Other containers should be left active in the Hierarchy by default in consideration of StreamSDKOption components that get/set PlayerPrefs and default values.



## StreamSDKOptionDropdown and StreamSDKOptionSlider

The StreamSDKOptionDropdown and StreamSDKOptionSlider components allow any UnityEngine.UI Dropdown or Slider element to automatically get/set its value with PlayerPrefs. In addition, it can route that value to the appropriate component and method. In StreamSDK 3.0, we have replaced the StreamSDKQualityManger and StreamSDKOptionsManager with this new system.

The way it works is that default values are set directly with UI elements (currently Dropdown and Slider are the only ones used). When the app runs, these default values will be set to PlayerPrefs under the name of the game object containing the StreamSDKOptionDropdown or StreamSDKOptionSlider component.

For example:

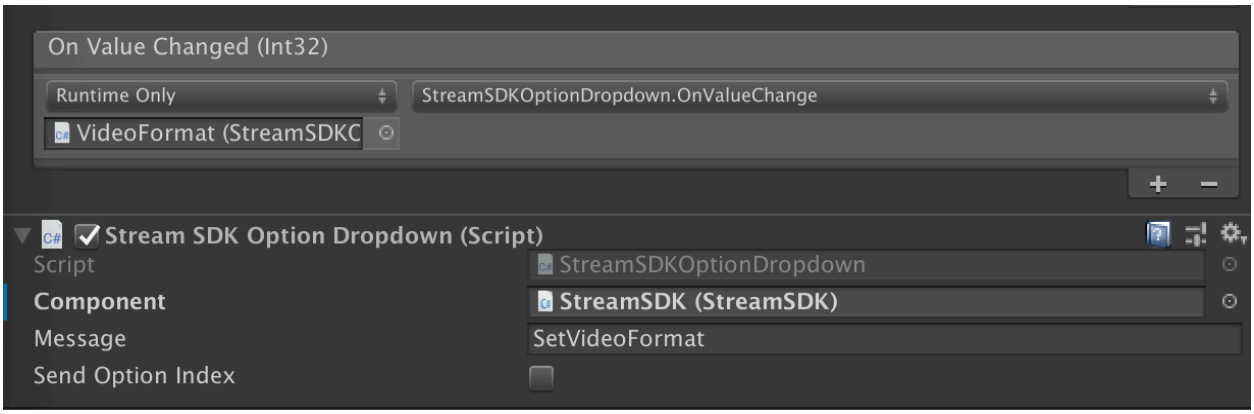
Hierarchy “Shell / MenuCanvas / OptionsAudio / AudioFrequency” will call “Set” + “AudioFrequency” (the name of the game object) AKA - StreamSDK.instance.SetAudioFrequency( string value ). Value in this case will be the string value contained within the Options array of the Dropdown component assigned to Hierarchy “Shell / MenuCanvas / OptionsAudio / AudioFrequency” i.e. 8000, 11025, 12000, 16000, 22050, 24000, 44100, 48000, 96000, or 192000.

When the Dropdown or Slider are enabled, their OnValueChanged() callback is automatically assigned to a StreamSDKOptionDropdown or StreamSDKOptionSlider OnValueChanged() method. Inside that method, the assigned component will be sent the specified message.

This system allows options to be setup in a very decentralized fashion without the overhead of compiling everything into one management script, as was done previously. Now, any option can easily be added and its results will be saved and routed to the appropriate place without writing any additional code.

*Note - with each of the StreamSDKOption* components, if no Component or Message is assigned, the component will default to StreamSDK and the message will default to “Set” + GameObject.name to which the StreamSDKOption* component is assigned to. This way, the component can just be dragged and dropped onto any Dropdown or Slider such as VideoFormat and OnValueChanged() will automatically send “SetVideoFormat” to StreamSDK.

## StreamSDKOptionDropdown



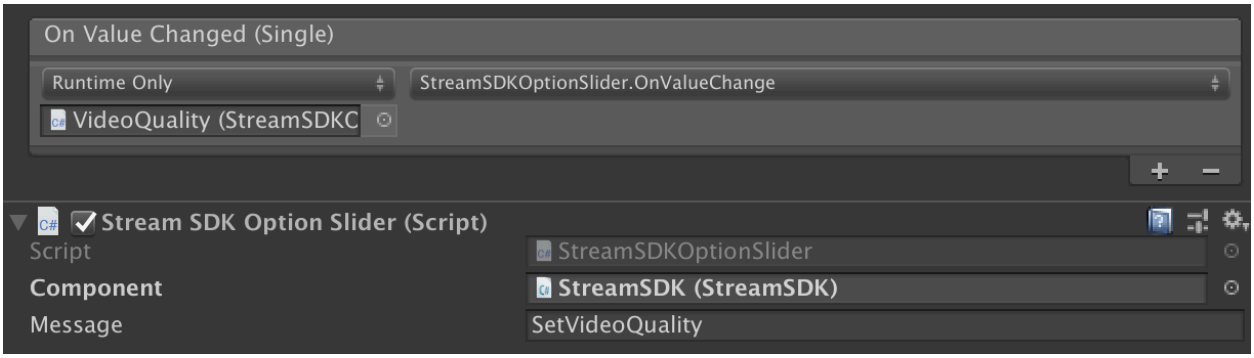

**Component -** This can be any component in the Scene to which a message may be relayed. If this is left blank it will default to the StreamSDK component in the Hierarchy.

**Message -** A string representing the name of a method to call on the assigned component. If this is left blank it will be assigned as “Set” + GameObject.name (the game object on which this component is assigned).

**Send Option Index** - A bool indicating if the option index of the currently selected Dropdown menu should be sent instead of the string value displayed in the Dropdown Options array. By default, the string or text in the Dropdown Options array is sent, but in certain instances the option index is preferable, usually when dealing with Enums.

## StreamSDKOptionSlider

The StreamSDKOptionSlider component is very similar to the StreamSDKDropdown component, however it does not have the Send Option Index field. Sliders always get/set their value directly.




**Component -** This can be any component in the scene to which a message may be relayed. If this is left blank it will default to the StreamSDK component in the Hierarchy.

**Message -** A string representing the name of a method to call on the assigned component. If this is left blank it will be assigned as “Set” + GameObject.name (referring to the game object on which this component is assigned).

## StreamSDK Getters (Network Transmission)

(generally, these yield data to be transmitted across the network)

**GetAudio**() - Gets the byte[] containing the local client's audio data.

**GetInput**() - Gets the byte[] containing the local client's input data.

**GetMouse**() - Gets the byte[] containing the local client's mouse data.

**GetStream**() - Gets the byte[] containing the local client's audio and video data.

**GetVideo**() - Gets the byte[] containing the local client's video data.

## StreamSDK Getters (Error)

StreamSDK has a number of backend related points of failure that developers must keep in mind to control the flow of User Experience (UX). If for example, the developer ships a StreamSDK app without an assigned API Key, an error will occur and the app will not process any streaming data. Use these boolean methods to keep tabs on the associated StreamSDK account and the UX of any associated apps.

*Use the StreamSDKErrorDebug prefab to print these errors on screen. Create a more polished integration to drive a custom user experience.

**GetErrorConnections()** - Returns true if the number of current connections exceeds the subscribed connection license.

**GetErrorCredentials()** - Returns true if there was a problem with the API Key or API Secret.

**GetErrorNoInternet()** - Returns true if StreamSDK cannot reach the Internet. This will occur if the app either has no license data or if it's been too long since it has checked-in with the StreamSDK servers. In general, the first time a StreamSDK app is run, it will need to check-in. After that, the app can use locally stored license data to run in case the app is being used in a LAN (local area network) situation or in case the StreamSDK servers are down. However, beyond a certain point, the app will need to check-in again. If the developer subscribes to additional license features, a check-in will need to be performed to verify those new features.

**GetErrorPlatform()** - Returns true if the current runtime platform does not have an associated platform license via the API Key.

**GetErrorVersionDeprecated()** - Returns true if the version of StreamSDK is deprecated.

**GetErrorVersionObsolete()** - Returns true if the version of StreamSDK is obsolete. No streaming will be processed in this case.

## StreamSDK Getters (Input)

StreamSDK allows input axes to be routed from one client to another. To access the input of a remote client, that client must be sending input via the StreamSDKTransporter > SendData > Input bool. It can be retrieved using the following StreamSDK getters.

**GetAxis( string name, int index )** - Returns the floating point value of the axis name using the index of the client. The index is relative to the local client in the order in which other clients connected. For example, if the local client just joined the master client, then the master client would be index 0 on the local client.

**GetAxisDown( string name, int index )** - Returns a bool as to whether or not the axis with name was just pressed down on client at index. The index is relative to the local client in the order in which other clients connected. For example, if the local client just joined the master client, then the master client would be index 0 on the local client.

## StreamSDK Getters (Mouse)

StreamSDK allows mouse input to be routed from one client to another. To access the mouse of a remote client, that client must be sending mouse data via the StreamSDKTransporter > SendData > Mouse bool. It can be retrieved using the following StreamSDK getters.

**GetMouseRemote( int index )** - Returns the mouse state of the client at index. The index is relative to the local client in the order in which other clients connected. For example, if the local client just joined the master client, then the master client would be index 0 on the local client.

**id** - int of the StreamSDK.instance.id of this remote mouse
**x** - float of the normalized x position of this remote mouse
**y** - float of the normalized y position of this remote mouse
**deltaX** - float of the last delta x value of this remote mouse
**deltaY** - float of the last delta y value of this remote mouse
**scroll** - float of the last scroll wheel value of this remote mouse
**lmb** - bool, is the left mouse button held on this remote mouse
**lmbDown** - bool, was the left mouse button clicked on this remote mouse
**lmbUp** - bool, was the left mouse button released on this remote mouse
**rmb** - bool, is the right mouse button held on this remote mouse
**rmbDown** - bool, was the right mouse button clicked on this remote mouse
**rmbUp** - bool, was the right mouse button released on this remote mouse
**mmb** - bool, is the middle mouse button held on this remote mouse
**mmbDown** - bool, was the middle mouse button clicked on this remote mouse
**mmbUp** - bool, was the middle mouse button released on this remote mouse

## StreamSDK Getters (Usage)

Each StreamSDK API Key has an allotted number of connections that can be associated with it. The number of allotted connections is upgradeable. To keep tabs on app usage, use these getters to inform your own backed so you can keep your StreamSDK account up to the demand of your app.

*Use the StreamSDKUsageDebug prefab to print these errors on screen. Create a more polished integration to drive a custom user experience.

**GetUserConnectionsCap()** - Returns an int representing the current cap or license limit of the account.

**GetUserConnectionsCurrent()** - Returns an int representing the current connections that have occurred during the previous rolling 30 days. If this number exceeds the current cap or license limit, the account should be upgraded immediately or app users will not have streaming capabilities.

**GetUserConnectionsPeak()** - Returns an int representing the peak connections that have ever occurred during any rolling 30 day period. If this number is close to exceeding the current cap or license limit, an upgrade should be considered.

## StreamSDK Setters

(for use with StreamSDKOptionDropdown and StreamSDKOptionSlider components or any extensions)

**SetAudioBufferSize**( string pAudioBufferSize ) - Sets StreamSDK.audioBufferSize accepts a string that is converted internally representing the buffer size i.e. “1” or “2”. A “1” is good for quicker latency but the possibility of buffer under-run. A “2” offers increased latency but reduced possibility of buffer under-run. Higher values than 2 are not recommended. It is also recommended to use SetDSPBufferSize() with an optimal value for the platform that is streaming instead of using this with a value other than 1, if possible.

**SetAudioCompressions**( int pAudioCompressions ) - Sets the compression cycles for StreamSDK audio. A “0” will not compress the audio at all, which is more performant at the risk of network bandwidth consumption. Higher values can reduce network bandwidth consumption.

**SetAudioCompressions**( string pAudioCompressions ) - Sets the compression cycles for StreamSDK audio. A “0” will not compress the audio at all, which is more performant at the risk of network bandwidth consumption. Higher values can reduce network bandwidth consumption.

**SetAudioFrequency**( int pAudioFrequency ) - Sets StreamSDK.audioFrequency and calls InitAudio( int numberReceivers ), accepts an int directly representing the frequency i.e. 8000, 11025, 12000, 16000, 22050, 24000, 44100, 48000, 96000, 192000.

**SetAudioFrequency**( string pAudioFrequency ) - Sets StreamSDK.audioFrequency and calls InitAudio( int numberReceivers ), accepts a string that is converted internally representing the frequency i.e. “8000”, “11025”, “12000”, “16000”, “22050”, “24000”, “44100”, “48000”, “96000”, “192000”. This makes it easy to create a Dropdown menu containing the frequencies that can call this method when OnValueChanged() is triggered.

**SetAudioType**( string pAudioType ) - Sets SteamSDK.audioType. Accepts a string that is converted ToLowerInvariant() representing the desired AudioType enum name i.e. “microphone”, “microphonemute”, “game”, “gamemute”, “none”, “nonemute”.“Microphone” etc. are acceptable as the strings are converted ToLowerInvariant() internally.

**SetCloneDepth**( string pCloneDepth ) - Sets StreamSDK.cloneDepth, accepts a string that is converted internally.
**SetCloneDepth**( string pCloneLayer ) - Sets StreamSDK.cloneLayer, accepts a string that is converted internally.
**SetCloneRect**( string pCloneRect ) - Sets StreamSDK.cloneRect, accepts a space-delimited string that is converted internally to Rect. For example “0 0 1 1” (x, y, width, height).
**SetDisplayType**( string pDisplayType ) - Sets StreamSDK.displayType, accepts a string that is converted internally ToLowerInvariant() (“native”, “clone”, “blind” ). “Native” etc. are acceptable as the strings are converted ToLowerInvariant() internally.
**SetDSPBufferSize**( float pFramerate ) - Sets StreamSDK.streamSDKAdvancedOptions.dspBufferSize. Sets Unity’s audioConfiguration.dspBufferSize. Accepts an int.
**SetFramerate**( float pFramerate ) - Sets StreamSDK.framerate. Accepts a float.
**SetFramerate**( string pFramerate ) - Sets StreamSDK.framerate. Accepts a string that is converted internally.
**SetGraphicsType**( string pGraphicsType ) - Sets StreamSDK.graphicsType. Accepts a string that is converted internally ToLowerInvariant() (“ar”, “game2d”, “game3d”, “videochat”, “vr”). “Game3D” etc. are acceptable as the strings are converted ToLowerInvariant().
**SetMicFrequency**( int pMicFrequency ) - Sets StreamSDK.micFrequency and calls InitMic( int numberReceivers ), accepts an int.
**SetMicFrequency**( int pMicFrequency ) - Sets StreamSDK.micFrequency and calls InitMic( int numberReceivers ), accepts a string that is converted internally.
**SetMicSensitivity**( int pMicSensitivity ) - Sets StreamSDK.micThreshold  Accepts a float.
**SetMicVolume**( float pMicVolume ) - Sets StreamSDK.micDecay  Accepts a float.
**SetMicVolume**( string pMicVolume ) - Sets StreamSDK.micDecay  Accepts a string that is converted internally.
**SetVideoCompressions**( int pVideoCompressions ) - Sets the compression cycles for StreamSDK video. A “0” will not compress the video at all, which is more performant at the risk of network bandwidth consumption. Higher values can reduce network bandwidth consumption.

**SetVideoCompressions**( string pVideoCompressions ) - Sets the compression cycles for StreamSDK video. A “0” will not compress the video at all, which is more performant at the risk of network bandwidth consumption. Higher values can reduce network bandwidth consumption.
**SetVideoFormat**( string pVideoFormat ) - Sets StreamSDK.width and StreamSDK.height. Accepts a string that is converted internally ToLowerInvariant() (“90p”, “180p”, “360p”, “720p”). “720P” etc. are acceptable as the strings are converted ToLowerInvariant().
**SetVideoFormatPrecise**( int width, int height ) - Sets StreamSDK.width and StreamSDK.height. Accepts two ints. Calls InitVideo();
**SetStreamCamera**( string pCameraName ) - Sets StreamSDK.streamCamera Accepts a string representing the GameObject in the scene holding the desired Camera component.
**SetStreamDisplays**( string pStreamDisplays ) - Sets StreamSDK.streamDisplays. Accepts a space-delimited string containing the names of the GameObjects holding the desired RawImage components i.e. “StreamDisplay1 StreamDisplay2 StreamDisplay3 StreamDisplay4”.
**SetVideoQuality**( int pVideoQuality ) - Sets StreamSDK.videoQuality  Accepts an int (1-100).
**SetVideoQuality**( float pVideoQuality ) - Sets StreamSDK.videoQuality  Accepts float (1-100).
**SetVideoQuality**( string pVideoQuality ) - Sets StreamSDK.videoQuality  Accepts a string that is converted internally i.e. “20”.

## Setting up with Photon

1. Create new Unity Project
2. Import StreamSDK package.
    a. You'll notice an error message the 'Photonview' could not be found. Let's fix that!
3. Import Photon.
    a. There are many networking services that are compatible with Unity, this demo uses Photon. You're welcome to use whichever is best for your project.
    b. Open the Asset Store.
    c. Search "Photon".
    d. Download and import “Photon Unity Networking Classic - Free" (it may be a few items down the list).
4. Set up Photon.
    a. Once imported, a window should pop up that says "PUN Setup".
    b. If you can't find this window, select Window > Photon Unity Networking > PUN Wizard > Setup Project.
    c. Enter your AppId if you already have an existing account.
    d. If you don't already have a Photon account, follow the instructions in the window to set one up at no charge.
    e. Enter AppId, select "Setup Project", close the window.
    f. If you’re working in Unity 2019.3 and above, you might need to delete the “Assets / Photon Unity Networking / Demos” folder.
5. Edit place-holder code in PhotonWrapper.cs.
    a. Open up the “StreamSDKVideoChat” scene from the Project View if you haven't already (“Assets / StreamSDK / Demo / VideoChat / VideoChat” ).
    b. If you try to play the scene, you'll notice one new error message.
    c. Double click that error message to open up the PhotonWrapper.cs script or search "PhotonWrapper" in the project view.
    d. Comment out the top line of PhotonWrapper.cs (#define submit should be //#define submit in order to use the full functionality of the class).
    e. Save the script and return to Unity.
6. Test StreamSDK locally before building.

   a. The demo scene should be working properly and you can play it.
   b. The first two buttons, “Create Video Chat” and “Join Video Chat”, will not work until another device has a build of this scene.
   c. Try the last button, “Test Video Chat”, to perform a round trip test stream locally.
   d. Also try the third button, “Calibrate Mic”, to prevent echo while still allowing other users to hear your microphone input (See the section “Calibrating the Mic" for more details.
7. Test StreamSDK over a network.
   a. You will now have to build out the demo scene and send the playable file to another device.
   b. File > Build Settings.
   c. If Photon automatically added scenes to the build list, remove them (select all, then press delete).
   d. Add the current StreamSDKVideoChat demo scene to the build settings (either click "Add Open Scenes" if it is currently open, or drag the scene over from the Project view).
   e. Select the target device you want to build for and click Build.
   f. Share that build with another device (email a zip, over FTP, over a USB, etc.).
   g. Open up the build on each device.
   h. Have one device "Create Video Chat" and have the other "Join Video Chat" after the first created the chat room.
   i. Now each device should be connected and streaming with one another.
   j. Be sure to exit and adjust mic settings if you are experiencing echo or can't hear each other.

## Calibrating the Mic

- While testing StreamSDK, you may notice an echo in the audio. This occurs because all devices have different mic sensitivity. StreamSDK is built to reduce this as much as possible, but sometimes users must adjust the mic sensitivity manually to reduce echo.
- In one of the StreamSDK Demo scenes, open the "Calibrate Mic" menu. Here, a single user can test their own mic and adjust sensitivity.
- The top slider allows the user to adjust volume level.
- The lower slider allows the user to adjust the mic sensitivity. The further to the left, the more sound the mic picks up. The further to the right, the more audio is blocked out. If the slider is too far left, the mic will be very sensitive and cause echo. If the slider is too far right, remote users will not be able to hear the local user.
- Adjust the slider on the lower bar until you can hear yourself well without echo.
- Test this by speaking and listening. You can also watch the visualizer in the bottom center. If it goes above the echo line while you speak normally, you may need to lower the sensitivity (move the Mic Sensitivity slider to the right). If it never goes above the drop line, you may need to raise sensitivity (move the Mic Sensitivity slider to the left).
- There should only ever be one AudioListener in your scene and it should be assigned to the StreamSDK game object.

## Integrating StreamSDK into your Project

- The resolution for all demos that come with StreamSDK is 1280x720 windowed as set in StreamSDKUIManagerScreen.SetResolution( 1280, 720, false ); This can be adjusted for individual projects, but it will require digging into StreamSDKUIManager.
- WebGL builds that are not set to 1280 x 720 (or whatever is set in StreamSDKUIManager) in the PlayerSettings by default will not render properly, nor will input register with the visuals if the PlayerSettings is mismatched with StreamSDKUIManager.
- All StreamSDK features should be adjustable on-the-fly so you can write code to dynamically adjust your stream to any scenario (bandwidth restricted, highest framerate, best quality, etc).
- StreamSDK allows developers to also set the quality of the various audio streams. audioFrequency determines the quality of the audio recorded by the app/game, while micFrequency controls the frequency recorded by the microphone. By default, StreamSDK uses 8KHz audio, which is telephone quality. StreamSDK will automatically resample the audio to whatever is specified by the developer (or in the options menu of the demos). However, not every device natively supports every audio frequency. If StreamSDK is forced to resample the audio, the quality of the audio will be altered.
- WebGL only supports 44100 audio reception at this time. This means that senders must use 44100 audioFrequency when they intend to send data to WebGL receivers. StreamSDK provides both downsampling and upsampling support in real-time. However, if possible using Project Settings > Audio > System Sample Rate set to 44100 in addition to setting the same frequency via the options menu in StreamSDK, will provide a higher quality sound. Some devices (Android for instance) won't recognize Project Settings > Audio > System Sample Rate. In these instances, StreamSDK will resample the audio from whatever the native device provides to the requested frequency in the options menu but the quality will be altered.
- WebGL builds in Unity are not capable of microphone input. Furthermore, AudioType.microphone and AudioType.microphonemute cannot be used to send

audio from the microphone of a non-WebGL sender to a WebGL receiver. However, StreamSDK can send mic input from a non-WebGL sender to WebGL using AudioType.micgamemute.

- WebGL builds in Unity are not capable of recording game audio.
- WebGL will crash on scenes that use StreamSDKTransporter.reloadScene, so it is advised to design scenes that do not require reloading.
- StreamSDKTransporter.numberReceivers should be set to the total number of clients in the network. For example, in a 1-to-1 video chat, numberReceivers should be 2.
- Example scenes make use of multiple cameras using Culling Masks and various Depths to show UI, Content, StreamDisplays etc. Three additional Layers called “Clone”, “StreamDisplay”, and “ContentCanvas” should be added to your project before importing StreamSDK.
- There should only ever be one AudioListener in your scene and it should be assigned to the StreamSDK game object.

## Creating the Base3D Scene from Scratch (almost)

The Base3D scene was created internally to test the steps necessary to go from blank scene to streaming with StreamSDK 3.0. Over the years, StreamSDK has had ebbs and flows of complexity with regard to its UX (or DX for Developer Experience). Version 3.0 has grown more complex than its predecessor, hopefully for the better. Here is a breakdown of steps necessary to create the Base3D scene using just a few prefabs.

*Note - To skip this, just drag the “StreamSDK / Demo / Base3D / Base3D” prefab into a scene. At that point, the content could just be edited to create any streaming experience. This is highly recommend, but the steps below might explain some of the details of the default Hierarchy structure for StreamSDK demos.

## Setup

1. **Main Menu, create a new scene:**
   File > New Scene.
2. **Hierarchy, deselect all, right click:**
   > Create Empty (do this 3x)
3. **Name 3 empty game objects:**
   “Application”
   “Content”
   “Shell”
4. **Hierarchy, multi-select:**
   “Application”
   “Content”
   “Shell”
5. **Inspector, set component values:**
   Transform {
   Position = (0, 0, 0)
   }
6. **Project View, drag prefab into the Hierarchy:**
   **Prefab** = “Assets / StreamSDK / Tool / Application / ApplicationTargetFramerate”
   **Target** = “Application"

7. **Project View, drag prefab into the Hierarchy:**
   **Prefab** = “Assets / StreamSDK / Tool / Application / QuitOnEscape”
   **Target** = “Application”

8. **Project View, drag prefab into Hierarchy:**
   **Prefab** = “StreamSDK / Core / StreamSDK”
   **Target** = “”

9. **Project View, drag prefab into Hierarchy:**
   **Prefab** = “Assets / StreamSDK / Tool / Transporter / StreamSDKTransporter”
   **Target** = “StreamSDK”

10. **Hierarchy, deselect all, right click:**
    > Create Empty

11. **Name new game object:**
    “CameraPivot”

12. **Hierarchy, select:**
    “CameraPivot”

13. **Inspector, set component values:**
    Transform {
        Position = (0, 0, 0)
    }

14. **Hierarchy, drag:**
    “CameraPivot” **—>** “Content”

15. **Hierarchy, deselect all, right click:**
    > Camera

16. **Name new game object:**
    “StreamCamera”

17. **Hierarchy, select:**
    “StreamCamera”

18. **Inspector, set component values:**
    Transform {
        Position = (0, 0, 0)
    }

19. **Hierarchy, drag game object:**
    “StreamCamera” **—>** “Content / CameraPivot”

20. **Hierarchy, select game object:**
    “Content / CameraPivot / StreamCamera”

21. **Inspector, set component values:**
    Camera {
        Culling Mask = Default

        Viewport Rect:
        X = 0, Y = 0
        W = 0.5, H = 1

        Depth = 2
    }

22. **Hierarchy, drag game object:**
    “Directional Light” **—>** “Content”

23. **Hierarchy, select game object:**
    “Content”

24. **Hierarchy, right click selected game object:**
    > Create > 3D Object > Sphere

25. **Hierarchy, select:**
    “Content / Sphere”

26. **Inspector, set component values:**
    Transform {
        Position = (3, 0, 0)
    }

27. **Hierarchy, select game object:**
    “Content”

28. **Right click selected game object:**
    > Create > 3D Object > Cube

29. **Hierarchy, select:**
    “Content / Cube”

30. **Inspector, set component values:**
    Transform {
        Position = (0, 0, 3)
    }

31. **Hierarchy, select game object:**
    “Content”

32. **Right click selected game object:**
    > Create > 3D Object > Capsule

33. **Hierarchy, select:**
    “Content / Capsule”

34. **Inspector, set component values:**
    Transform {
        Position = (0, 0, -3)
    }

35. **Hierarchy, select game object:**
    “Content”

36. **Right click selected game object:**
    > Create > 3D Object > Cylinder

37. **Hierarchy, select:**
    “Content / Cylinder”

38. **Inspector, set component values:**
    Transform {
        Position = (-3, 0, 0)
    }

39. **Hierarchy, deselect all, right click:**
    UI > Canvas.

40. **Name new game object:**
    “StreamDisplayCanvas”

41. **Hierarchy, select:**
    “StreamDisplayCanvas”

42. **Inspector, set Layer:**
    StreamDisplay.

43. **Inspector, set component values:**
    Canvas {
        Render Mode = World Space

        Rect Transform:
        Pos X = 0
        Pos Y = 0
        Width = 1280
        Height = 720
        Pivot = (0.5, 0.5)
        Scale = (1, 1, 1)
    }

44. **Hierarchy, drag:**
    “StreamDisplayCanvas” **—>** “Content”.

45. **Hierarchy, deselect all, right click:**
    > Camera

46. **Name new game object:**
    “StreamDisplayCamera”

47. **Hierarchy, select:**
    “StreamDisplayCamera”

48. **Inspector, set component values:**
    Transform {
        Position = (0, 0, -1)
    }

49. **Hierarchy, drag:**
    “StreamDisplayCamera” **—>** “Content / StreamDisplayCanvas”

50. **Hierarchy, select:**
    “Content / StreamDisplayCanvas / StreamDisplayCamera”

51. **Inspector, set component values:**
    Camera {
        Culling Mask = StreamDisplay
        Projection = Orthographic
        Size = 360

        Viewport Rect {
            X = 0.5, Y = 0
            W = 0.5, H = 1.
        }

      Depth = 3
    }

52. **Project View, drag prefab to Hierarchy:**
    Prefab = “Assets / StreamSDK / Tool / StreamDisplay / StreamDisplay”
    Target = “Content / StreamDisplayCanvas”

53. **Hierarchy, select:**
    “Content / StreamDisplayCanvas”

54. **Inspector, set component values:**
    Canvas {
        Event Camera = “Content / StreamDisplayCanvas / StreamDisplayCamera”
    }

55. **Hierarchy, deselect all, right click:**
    UI > Canvas.

56. **Name new game object:**
    “ContentCanvas”.

57. **Hierarchy, select:**
    “ContentCanvas”.

58. **Inspector, set component values:**
    Canvas {
        Render Mode = World Space
    }

Rect Transform {
    Pos X = 0
    Pos Y = 0
    Width = 1280
    Height = 720
}

59. **Hierarchy, deselect all, right click:**
    > Camera.

60. **Name the new game object:**
    “ContentCanvasCamera”

61. **Inspector, set component values:**
    Transform {
        Position = (0, 0, -1)
    }

    Camera {
        Clear Flags = Don’t Clear
        Culling Mask = UI
        Projection = Orthographic
        Size = 360
        Depth = 10
    }

62. **Hierarchy, drag:**
    “ContentCanvasCamera” **—>** “ContentCanvas”.

63. **Hierarchy, select:**
    “ConventCanvas”

64. **Inspector, set component values:**
    Canvas {
        Event Camera = “Content / ContentCanvas / ContentCanvasCamera”
    }

65. **Hierarchy, select:**
    “ContentCanvas”.

66. **Hierarchy, right click:**
    UI > Button

67. **Name the new game object:**
    “BackButton”.

68. **Hierarchy, select:**
    “Content / ContentCanvas / BackButton”

69. **Inspector, set Layer:**
    UI

70. **Inspector, set component values:**
    Rect Transform {
        Anchor = bottom center
        Pos X = 0
        Pos Y = 0
        Width = 1280
        Height = 40
        Anchors
        Min = (0.5, 0)
        Max = (0.5, 0)
        Pivot = (0.5, 0)
    }

71. **Hierarchy, select:**
    “ContentCanvas / BackButton / Text”

72. **Inspector set component values:**
    Text {
        Text = “Back”
        Best Fit = true
        Min Size = 10
        Max Size = 40
    }

73. **Hierarchy, select:**
    “ContentCanvas / BackButton”

74. **Inspector, add OnClick() event:**
    **Object** = “StreamSDK / StreamSDKTransporter”
    **Function** = StreamSDKTransporter.QuitServer()

75. **Hierarchy, drag:**
    “ContentCanvas” **—>** “Content”

76. **Project View, drag a prefab to Hierarchy:**
    **Prefab** = “Assets / StreamSDK / Tool / Shell / UIManager / StreamSDKUIManager”
    **Target** = “Shell”

77. **Hierarchy, rename:**
    “Main Camera” >< “MenuCamera”.

78. **Hierarchy, deselect all, right click:**
    UI > Canvas

79. **Name new game object:**
    “MenuCanvas”.

80. **Hierarchy, select:**
    “MenuCanvas”

81. **Inspector, set component values:**
    Canvas {
        Render Mode = World Space
    }

    Rect Transform {
        Pos X = 0
        Pos Y = 0
        Width = 1280
        Height = 720
    }

82. **Hierarchy, drag:**
    “MenuCanvas” **—>** “Shell”

83. **Hierarchy, drag:**
    “MenuCamera” —> “Shell / MenuCanvas”

84. **Hierarchy, select:**
    “Shell / MenuCanvas”

85. **Inspector, set component values:**
    Canvas {
        Event Camera = “Shell / MenuCanvas / MenuCamera”
    }

86. **Hierarchy, deselect all, right click:**
    UI > EventSystem
    (If there is already one in the scene, skip this step and use the previously existing one instead).

87. **Hierarchy, drag:**
    “EventSystem” **—>** “Shell / MenuCanvas”

88. **ProjectView, drag to Hierarchy:**
    **Prefab** = “Assets / StreamSDK / Tool / Shell / CalibrateMic / CalibrateMic”
    **Target** = “Shell / MenuCanvas”

89. **ProjectView, drag to Hierarchy:**
    **Prefab** = “Assets / StreamSDK / Tool / Shell / Options / Options”
    **Target** = “Shell / MenuCanvas”

90. **ProjectView, drag to Hierarchy:**
    **Prefab** = “Assets / StreamSDK / Tool / Shell / Options / OptionsAudio”
    **Target** = “Shell / MenuCanvas”

91. **ProjectView, drag to Hierarchy:**
    **Prefab** = “Assets / StreamSDK / Tool / Shell / Options / OptionsNetwork”
    **Target** = “Shell / MenuCanvas”

92. **ProjectView, drag to Hierarchy:**
    **Prefab** = “Assets / StreamSDK / Tool / Shell / Options / OptionsVideo”
    **Target** = “Shell / MenuCanvas”

93. **ProjectView, drag to Hierarchy:**
    **Prefab** = “Assets / StreamSDK / Tool / Shell / MainMenu / MainMenu”
    **Target** = “MenuCanvas”

94. **Hierarchy, deselect all, right click:**
    > Camera

95. **Name new game object:**
    “BackgroundCamera”

96. **Hierarchy, select:**
    “BackgroundCamera”

97. **Inspector, set component values:**
    Transform {
        Position = (0, 0, 0)
    }

98. **Hierarchy, drag:**
    “BackgroundCamera” **—>** “Shell / MenuCanvas”

99. **Hierarchy, select:**
    “Shell / MenuCanvas / BackgroundCamera”

100. **Inspector, set component values:**

    Camera {
        Clear Flags = Solid Color.
        Background = Black
        Culling Mask = Nothing
        Depth = 99
    }

101. **Hierarchy, select:**
    “Shell / MenuCanvas / MenuCamera”

102. **Inspector, set component values:**
    Transform {
        Position = ( 0, 0, -1 )
    }

    Camera {
        Clear Flags = Don’t Clear
        Culling Mask = UI
        Projection = Orthographic
        Size = 360

Depth = 1
}

103. **Hierarchy, select:**
“Content / CameraPivot / StreamCamera”

104. **Inspector, set component values:**
Transform {
Position = ( 0, 0, 0 )
}

105. **Hierarchy, select:**
“Shell / MenuCanvas / MainMenu / Options”

106. **Inspector, add OnClick() event:**
**Object** = “Shell / StreamSDKUIManager”
**Function** = StreamSDKUIManager.ShowContainer(GameObject)
**GameObject** = “Shell / MenuCanvas / Options”

107. **Hierarchy, select:**
“Shell / MenuCanvas / MainMenu / CalibrateMic”

108. **Inspector add OnClick() event:**
**Object** = “Shell / StreamSDKUIManager”
**Function** = StreamSDKUIManager.ShowContainer(GameObject)
**GameObject** = “Shell / MenuCanvas / CalibrateMic”

109. **Hierarchy, select:**
“Shell / MenuCanvas / MainMenu / Test”

110. **Inspector, add OnClick() event:**
**Object** = “StreamSDK / StreamSDKTransporter”
**Function** = StreamSDKTransporter.StartServer( bool )
**bool** = true

111. **Hierarchy, select:**
“Shell / MenuCanvas / MainMenu / Join”

112. **Inspector, add OnClick() event:**
**Object** = “StreamSDK / StreamSDKTransporter”.

**Function** = StreamSDKTransporter.JoinServer( bool )
**bool** = false

113. **Hierarchy, select:**
"Shell / Canvas / MainMenu / Create"

114. **Inspector, add OnClick() event:**
**Object** = "StreamSDK / StreamSDKTransporter"
**Function** = Assign StreamSDKTransporter.StartServer(bool)
**bool** = false

115. **Hierarchy, select:**
"Shell / MenuCanvas / OptionsNetwork / PhotonRegion"

116. **Inspector, set component values:**
StreamSDKOptionDropdown {
Component = "StreamSDK / StreamSDKTransporter"
Message = ""
Send Option Index = true
}

117. **Hierarchy, select:**
"Shell / MenuCanvas / OptionsNetwork / PhotonProtocol"

118. **Inspector, set component values:**
StreamSDKOptionDropdown {
Component = "StreamSDK / StreamSDKTransporter"
Message = ""
Send Option Index = true
}

119. **Hierarchy, select:**
"Shell / MenuCanvas / OptionsNetwork / IPAddressA"

120. **Inspector, set component values:**
StreamSDKOptionDropdown {
Component = "StreamSDK / StreamSDKTransporter"
Message = ""
Send Option Index = true
}

121. **Hierarchy, select:**
“Shell / MenuCanvas / OptionsNetwork / IPAddressB”

122. **Inspector, set component values:**
StreamSDKOptionDropdown {
Component = “StreamSDK / StreamSDKTransporter”
Message = “”
Send Option Index = true
}

123. **Hierarchy, select:**
“Shell / MenuCanvas / OptionsNetwork / IPAddressC”

124. **Inspector, set component values:**
StreamSDKOptionDropdown {
Component = “StreamSDK / StreamSDKTransporter”
Message = “”
Send Option Index = true
}

125. **Hierarchy, select:**
“Shell / MenuCanvas / OptionsNetwork / IPAddressC”

126. **Inspector, set component values:**
StreamSDKOptionDropdown {
Component = “StreamSDK / StreamSDKTransporter”
Message = “”
Send Option Index = true
}

127. **Hierarchy, select:**
“Shell / MenuCanvas / OptionsVideo **/** BackButton”

128. **Inspector add OnClick() event:**
**Object** = “Shell / StreamSDKUIManager”
**Function** = StreamSDKUIManager.ShowContainer(GameObject)
**GameObject** = “Shell / MenuCanvas / Options”

129. **Hierarchy, select:**
“Shell / MenuCanvas / OptionsNetwork **/** BackButton”

130. **Inspector add OnClick() event:**
    **Object** = “Shell / StreamSDKUIManager”
    **Function** = StreamSDKUIManager.ShowContainer(GameObject)
    **GameObject** = “Shell / MenuCanvas / Options”

131. **Hierarchy, select:**
    “Shell / MenuCanvas / OptionsAudio **/** BackButton”

132. **Hierarchy add OnClick() event:**
    **Object** = “Shell / StreamSDKUIManager”
    **Function** = StreamSDKUIManager.ShowContainer(GameObject)
    **GameObject** = “Shell / MenuCanvas / Options”

133. **Hierarchy select:**
    “Shell / MenuCanvas / Options **/** BackButton”

134. **Hierarchy add OnClick() event:**
    **Object** = “Shell / StreamSDKUIManager”
    **Function** = StreamSDKUIManager.ShowContainer(GameObject)
    **GameObject** = “Shell / MenuCanvas / MainMenu”

135. **Hierarchy select:**
    “Shell / MenuCanvas / CalibrateMic **/** BackButton”

136. **Hierarchy add OnClick() event:**
    **Object** = “Shell / StreamSDKUIManager”
    **Function** = StreamSDKUIManager.ShowContainer(GameObject)
    **GameObject** = “Shell / MenuCanvas / MainMenu”

137. **Hierarchy, select:**
    “Shell / Options / Audio”

138. **Inspector, add OnClick() event:**
    **Object** = StreamSDKUIManager
    **Function** = StreamSDKUIManager.ShowContainer(GameObject)
    **GameObject** = “Shell / MenuCanvas / OptionsAudio”

139. **Hierarchy, select:**
    “Shell / Options / Video”

140. **Inspector, add OnClick() event:**
    **Object** = StreamSDKUIManager
    **Function** = StreamSDKUIManager.ShowContainer(GameObject)
    **GameObject** = “Shell / MenuCanvas / OptionsVideo”

141. **Hierarchy, select:**
    “Shell / Options / Network”

142. **Inspector, add OnClick() event:**
    **Object** = StreamSDKUIManager
    **Function** = StreamSDKUIManager.ShowContainer(GameObject)
    **GameObject** = “Shell / MenuCanvas / OptionsNetwork”

143. **Hierarchy, select:**
    “Shell / StreamSDKUIManager”

144. **Inspector, set component values:**

```
StreamSDKUIManager {
    Containers {
        Size = 8
        Element 0 = “Shell / MenuCanvas / MainMenu”
        Element 1 = “Shell / MenuCanvas / OptionsVideo”
        Element 2 = “Shell / MenuCanvas / OptionsNetwork”
        Element 3 = “Shell / MenuCanvas / OptionsAudio”
        Element 4 = “Shell / MenuCanvas / Options”
        Element 5 = “Shell / MenuCanvas / CalibrateMic”
        Element 6 = “Content”
        Element 7 = “Shell / MenuCanvas / BackgroundCamera”
    }
}
```

    * Note - “MenuCanvas / MainMenu” should be first, as upon startup, Container [0] is shown and the others are hidden.

145. **Inspector, set component values:**

```
StreamSDKUIManager {
    Event System = “Shell / MenuCanvas / EventSystem”
}
```

* Note - If this is not assigned, StreamSDKUIManager will assign EventSystem.current automatically, but in scenes with more than one EventSystem this may not be desirable.

146. **Hierarchy, select:**
"StreamSDK / StreamSDKTransporter"

147. **Inspector, set component values:**
StreamSDKTransporter {
Content Container = "Content"
Ip Address = ""
Net Tech = Photon
Number Receivers = 2
Override Photon Server = false
Reload Scene = false
Room Name = Base3D
Send Rate = 50
Test = false
Send Audio = true
Send Video = true
Send Control = false
Send Mouse = false
}

148. **Hierarchy, select:**
"StreamSDK"

149. **Inspector, set component values:**
StreamSDK {
Api Key = Your API Key from StreamSDK.com
Api Secret = Your API Secret from StreamSDK.com
Stream Camera = "Content / CameraPivot / StreamCamera"
Stream Displays {
Size = 1
Element 0 = "Content / StreamDisplayCanvas / StreamDisplay"
}
}

150. **Hierarchy, select:**
"Content / CameraPivot"

151. **Inspector, Add Component:**
    TransformRotate

152. **Inspector set component values:**
    TransformRotate {
        Speed = 100
        Axes = (0,1,0)
        Space = World
    }

153. **Hierarchy, multi-select:**
    “Content / CameraPivot / StreamCamera”
    “Content / StreamDisplayCanvas / StreamDisplayCamera”
    “Content / ContentCanvas / ContentCanvasCamera”
    “Shell / MenuCanvas / MenuCamera”
    “Shell / MenuCanvas / BackgroundCamera”

154. **Inspector, right click on component:**
    AudioListener

155. **Right click menu, choose:**
    Remove Component

156. **Hierarchy, select:**
    “Shell / MenuCanvas / MainMenu / Create”

157. **Inspector, add additional OnClick() event:**
    **Object** = “Content / StreamDisplayCanvas”
    **Function** = GameObject.SetActive( false )

158. **Hierarchy, select:**
    “Shell” / MenuCanvas / MainMenu / Test”

159. **Inspector, add additional OnClick() event:**
    **Object** = “Content / StreamDisplayCanvas”
    **Function** = GameObject.SetActive( true )

160. **Hierarchy, select:**
    “Content”.

161. **Inspector, disable selected game object**

162. **Hierarchy, select:**
    “Shell / MenuCanvas”

163. **Inspector, set Layer:**
    UI (apply to children)

## Audio Optimization

StreamSDK provides a range of options to control audio flow and quality.

By default StreamSDK Advanced Options has a flag for resampleAudio that is set to true. This means that any audio processed for transmission will be resampled to the frequency set by StreamSDK.instance.SetAudioFrequency().

### Resampling with Multiples

Resampling has some benefits. First, bandwidth can be reduced, for example sending audio at 22,050 Hz will use less bandwidth than audio sent at 44,100 Hz. However, the quality will also be reduced. In the example of downsampling from 44,100 Hz to 22,050 Hz, the reduction in quality will be negligible since those frequencies are multiples of each other.

### Resampling non-Multiples

If downsampling is done from 44,100 Hz to 24,000 Hz the audio quality will be noticeably worse than downsampling to 22,050, since 44,100 and 24,000 are not multiples of each other. StreamSDK does some neat tricks to overcome this hurdle so the audio will sound as good as it can, while being processed in real-time, but it will not sound as good as resampling with multiples.

### Not Resampling

The best audio quality results from not resampling audio at runtime. This method requires that the audio clip and the system rate of the casting device be aligned. For example, if an audio clip is recorded at 48,000 Hz then StreamSDK's audio frequency needs to be set to 48,000 Hz on the casting device. Any receiving devices will automatically inherit the casting device's audio frequency. If the the audio clip and the casting device's audio frequency are misaligned, the audio will sound fine on the casting device but it will not sound right on receiving devices.

**AudioBufferSize and DSPBufferSize**

Depending on the devices involved in the network and the audio settings, there may be pauses in the audio. A quick solution is to use StreamSDK.instance.SetAudioBufferSize( 2 ) (or use the options menu provided in the Shell prefab). This will buffer audio on the casting device longer before sending occurs. Currently, this solution is non-optimal though and results in audio artifacts on the receiver.

A better solution is to find the optimal DSPBufferSize (using StreamSDK.instance.SetDSPBufferSize() ) for the casting device. In testing, lower values worked well on an old Android device while higher values worked well on more powerful laptop computers. This is relative to the casting device. By default, StreamSDK uses a DSPBufferSize of 256, which is considered “Low Latency” by Unity. For best results on more powerful systems a value of 2048 works well, while low-powered devices can fair well with values of 128 or even 64.

**WebGL**

Audio in WebGL is different than other platforms in Unity because it does not use FMOD. Therefore, it does not support the microphone or any audio recording at all. In fact, currently, just having an AudioListener in the scene will cause a WebGL build to freeze when run in a browser with StreamSDK. Fortunately, an AudioListener is not required to hear audio with WebGL and StreamSDK. Furthermore, only 44,100 Hz audio is supported in WebGL. The best recommendation as of version 3.2.0, when streaming to WebGL is 44,100 Hz audio, NOT resampled, with an AudioBufferSize of 2.